Javier Nieves & Nicholas Redmond

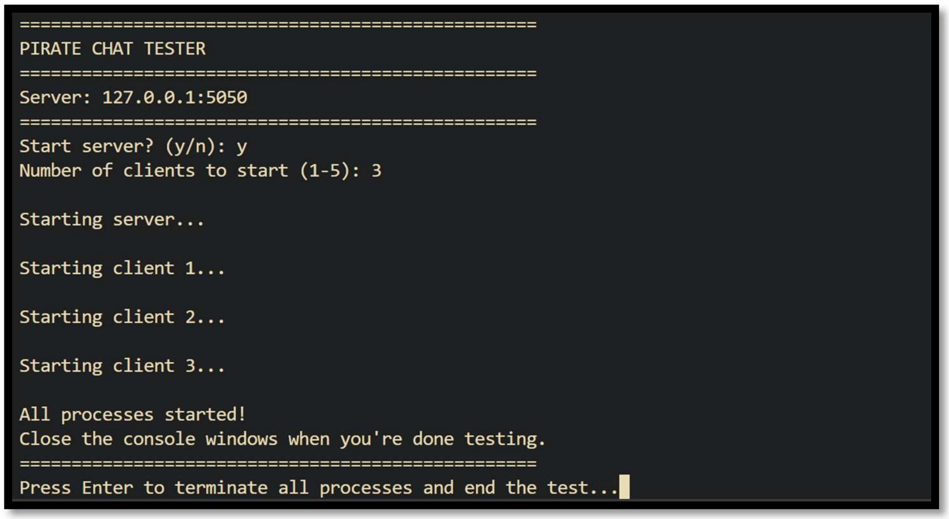
Professor Zhang

COMP 3825

24 April 2025

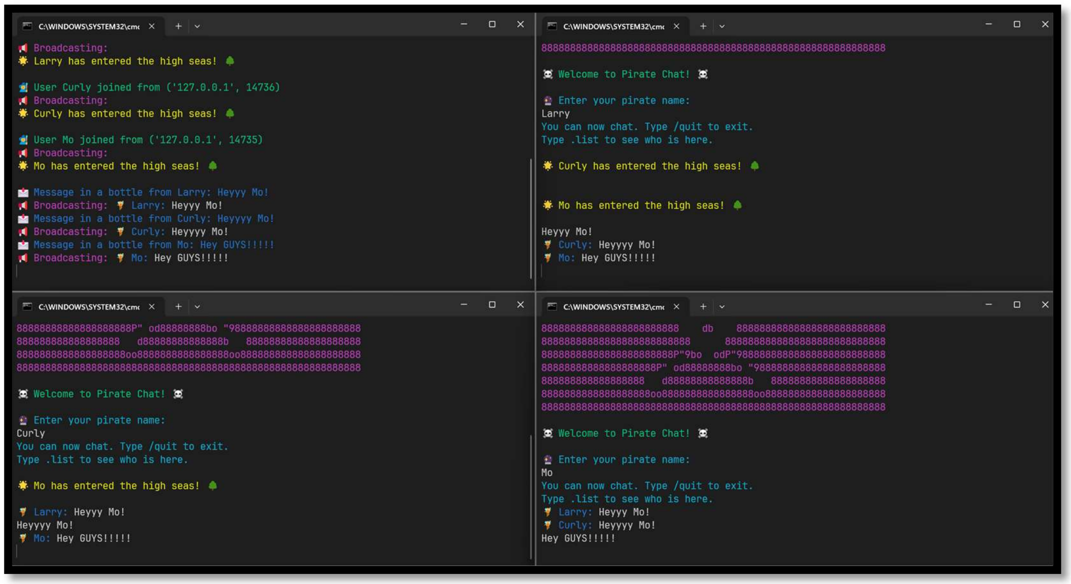
Pirate Chat

Pirate Chat is a secure, real-time messaging platform developed in the Python programming language, and it allows multi-client support, allowing various users to communicate in a private chat room environment. The application offers encrypted message exchange and concurrent client handling by leveraging the power of Python's socket, SSL (TLS/SSL wrapper for socket object), and threading libraries. Adding a playful twist with the pirate theme adds a thematic charm, while the implementation demonstrates proficiency in secure programming. This report will outline the design, functionality, and evaluation of the Pirate Chat system, offering an insight into its architecture, communication model, installation process, and performance.



```
==================================================
PIRATE CHAT TESTER
==================================================
Server: 127.0.0.1:5050
==================================================
Start server? (y/n): y
Number of clients to start (1-5): 3

Starting server...

Starting client 1...

Starting client 2...

Starting client 3...

All processes started!
Close the console windows when you're done testing.
==================================================
Press Enter to terminate all processes and end the test...
```

The design of Pirate Chat emphasizes security. On the server side of the application, the application would listen for a client's connection using an SSL wrapped socket, which is secured with a self-signed certificate and private key (server.crt and server.key). Once a connection is established, the server will be able to spawn a new thread to manage each client independently. This threading model ensures the server can manage multiple users simultaneously. Each client is tracked in a shared list that is safeguarded using threading locks, enabling synchronized access during message broadcasting. While on the client side, the users are connected by entering a unique pirate name, after which they receive a welcome message and a Jolly Roger ASCII banner. Clients are then able to send and receive messages in real time chat. The application also supports custom commands such as ".list", which provides a list of active users, and ".exit", which allows for a graceful disconnection. Messages sent by one client are relayed by the server to all others, excluding the sender, using a broadcast mechanism.



```
C:\WINDOWS\SYSTEM32\cmd
📢 Broadcasting:
🌟 Larry has entered the high seas! 🌳

🧑 User Curly joined from ('127.0.0.1', 14736)
📢 Broadcasting:
🌟 Curly has entered the high seas! 🌳

🧑 User Mo joined from ('127.0.0.1', 14735)
📢 Broadcasting:
🌟 Mo has entered the high seas! 🌳

📩 Message in a bottle from Larry: Heyyy Mo!
📢 Broadcasting: 🍺 Larry: Heyyy Mo!
📩 Message in a bottle from Curly: Heyyyy Mo!
📢 Broadcasting: 🍺 Curly: Heyyyy Mo!
📩 Message in a bottle from Mo: Hey GUYS!!!!!
📢 Broadcasting: 🍺 Mo: Hey GUYS!!!!!
```

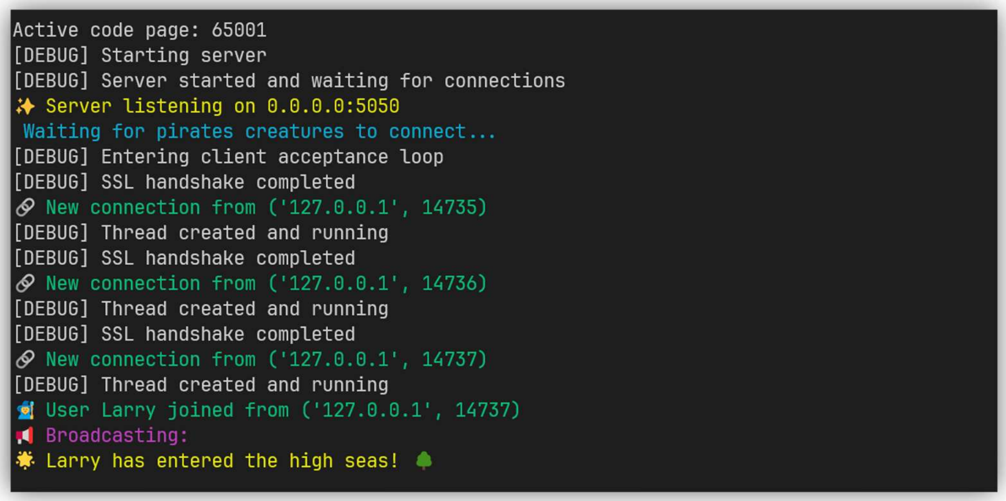
```
C:\WINDOWS\SYSTEM32\cmd
88888888888888888888888888888888888888888888888888888888888888

☠ Welcome to Pirate Chat! ☠

🔮 Enter your pirate name:
Larry
You can now chat. Type /quit to exit.
Type .list to see who is here.

🌟 Curly has entered the high seas! 🌳


🌟 Mo has entered the high seas! 🌳

Heyyy Mo!
🍺 Curly: Heyyyy Mo!
🍺 Mo: Hey GUYS!!!!!
```

```
C:\WINDOWS\SYSTEM32\cmd
8888888888888888888P" od8888888bo "9888888888888888888888
888888888888888888   d88888888888888b   88888888888888888888
8888888888888888888oo888888888888888oo8888888888888888888888
88888888888888888888888888888888888888888888888888888888888888

☠ Welcome to Pirate Chat! ☠

🔮 Enter your pirate name:
Curly
You can now chat. Type /quit to exit.
Type .list to see who is here.

🌟 Mo has entered the high seas! 🌳

🍺 Larry: Heyyy Mo!
Heyyyy Mo!
🍺 Mo: Hey GUYS!!!!!
```

```
C:\WINDOWS\SYSTEM32\cmd
888888888888888888888888    db    888888888888888888888888888
88888888888888888888888888      8888888888888888888888888888
8888888888888888888888P"9bo  odP"9888888888888888888888888888
8888888888888888888P" od8888888bo "9888888888888888888888
888888888888888888   d88888888888888b   88888888888888888888
8888888888888888888oo888888888888888oo8888888888888888888888
88888888888888888888888888888888888888888888888888888888888888

☠ Welcome to Pirate Chat! ☠

🔮 Enter your pirate name:
Mo
You can now chat. Type /quit to exit.
Type .list to see who is here.
🍺 Larry: Heyyy Mo!
🍺 Curly: Heyyyy Mo!
Hey GUYS!!!!!
```

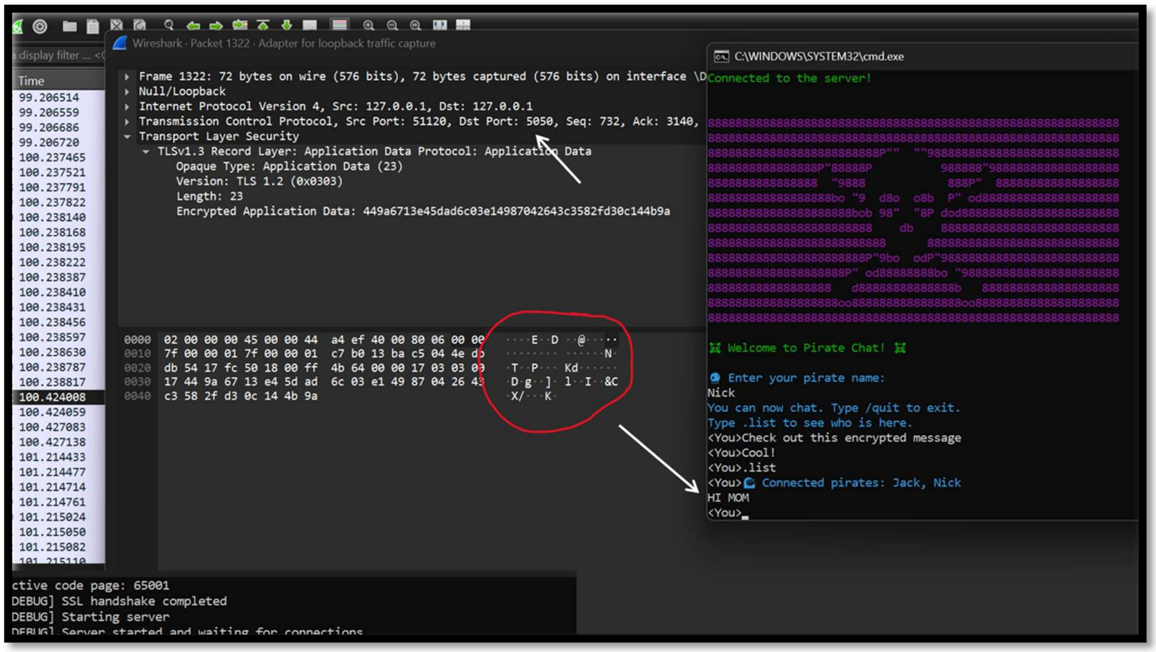
Communication between clients is routed entirely through the server. Each message received from a client is handled in its dedicated thread and then broadcast to other clients. Special commands are interpreted by the client application itself. For instance, when a user types ".list", the client requests the list of currently connected pirate names from the server. If a user enters ".exit", the client notifies the server of the disconnect, terminates its session, and the server updates the client list accordingly. This message flow illustrates the seamless communication and graceful exit strategy implemented within the chat environment. Following along to how you would be able to run Pirate Chat, users must ensure their system meets a few basic requirements. Python 3.x and OpenSSL must be installed to enable secure connections. The SSL certificate can be generated using the command:

openssl req -new -x509 -days 365 -nodes -out server.crt -keyout server.key.



```
Active code page: 65001
[DEBUG] Starting server
[DEBUG] Server started and waiting for connections
✨ Server listening on 0.0.0.0:5050
 Waiting for pirates creatures to connect...
[DEBUG] Entering client acceptance loop
[DEBUG] SSL handshake completed
🔗 New connection from ('127.0.0.1', 14735)
[DEBUG] Thread created and running
[DEBUG] SSL handshake completed
🔗 New connection from ('127.0.0.1', 14736)
[DEBUG] Thread created and running
[DEBUG] SSL handshake completed
🔗 New connection from ('127.0.0.1', 14737)
[DEBUG] Thread created and running
🧑 User Larry joined from ('127.0.0.1', 14737)
📢 Broadcasting:
🌞 Larry has entered the high seas! 🌳
```

Finally, the users will have the certificates ready, and the users can launch the server by running "pirate_server.py". Clients can then join by executing "pirate_client.py". Both files must be in the appropriate directories, and the server's certificate must be accessible to the client for trust validation. Some of the challenges we faced lay mostly in the SSL encryption process. It was very easy to verify the communication from socket to socket. Verifying that we had implemented the SSL encryption correctly was a bit trickier. Ultimately we used a packet-sniffer called Wireshark to capture the data packets send between client and server, to verify encryption. It indeed encrypted the data but also appeared to use an updated TCP protocol titled TCP 1.2 for transmission.


display filter ... <
Wireshark · Packet 1322 · Adapter for loopback traffic capture
C:\WINDOWS\SYSTEM32\cmd.exe

```
Time
99.206514
99.206559
99.206686
99.206720
100.237465
100.237521
100.237791
100.237822
100.238140
100.238168
100.238195
100.238222
100.238387
100.238410
100.238431
100.238456
100.238597
100.238630
100.238787
100.238817
100.424008
100.424059
100.427083
100.427138
101.214433
101.214477
101.214714
101.214761
101.215024
101.215050
101.215082
101.215110
```

```
Frame 1322: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface \D
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 51120, Dst Port: 5050, Seq: 732, Ack: 3140,
Transport Layer Security
  TLSv1.3 Record Layer: Application Data Protocol: Application Data
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 23
    Encrypted Application Data: 449a6713e45dad6c03e14987042643c3582fd30c144b9a
```

```
0000  02 00 00 00 45 00 00 44  a4 ef 40 00 80 06 00 00   ····E·D ··@·····
0010  7f 00 00 01 7f 00 00 01  c7 b0 13 ba c5 04 4e db   ········ ······N·
0020  db 54 17 fc 50 18 00 ff  4b 64 00 00 17 03 03 00   ·T··P··· Kd······
0030  17 44 9a 67 13 e4 5d ad  6c 03 e1 49 87 04 26 43   ·D·g··]· l··I··&C
0040  c3 58 2f d3 0c 14 4b 9a                            ·X/···K·
```

```
ctive code page: 65001
DEBUG] SSL handshake completed
DEBUG] Starting server
DEBUG] Server started and waiting for connections
```

```
Connected to the server!


8888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888P""  ""9888888888888888888888888888
8888888888888888P"88888P          988888"988888888888888888
88888888888888888  "9888            888P"  88888888888888888
8888888888888888888bo "9  d8o  o8b  P" od888888888888888888
88888888888888888888888bob 98"  "8P dod888888888888888888888
88888888888888888888888888    db    8888888888888888888888888
8888888888888888888888888888      8888888888888888888888888888
8888888888888888888888888P"9bo  odP"98888888888888888888888888
888888888888888888888P" od8888888bo "9888888888888888888888
88888888888888888888   d88888888888888b   888888888888888888
888888888888888888888oo8888888888888888oo888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888

☠ Welcome to Pirate Chat! ☠

🧭 Enter your pirate name:
Nick
You can now chat. Type /quit to exit.
Type .list to see who is here.
<You>Check out this encrypted message
<You>Cool!
<You>.list
<You>🌊 Connected pirates: Jack, Nick
HI MOM
<You>_
```

To illustrate its functionality, the application provides clear visual outputs in the terminal. The server window displays connection logs, SSL handshake messages, and broadcast logs. Meanwhile, the client terminal shows the welcome banner, message exchanges, and responses to commands such as “.list” and “.exit”. These screenshots, available in the accompanying appendix, visually confirm the smooth operation and encryption features of the application. From an evaluative standpoint, Pirate Chat excels in several areas. It achieves secure, real-time messaging with SSL encryption and concurrent processing through threads. The user interface, while basic, is clear and responsive, and command handling adds a layer of interactivity. Nonetheless, there are areas for future improvement. Introducing a graphical user interface (GUI) would enhance usability, while better certificate management, such as using a trusted Certificate Authority (CA), could improve accessibility. Additionally, integrating user authentication would help validate identities and further protect the chat environment. In conclusion, Pirate Chat is a well-rounded, educational application that demonstrates essential concepts in network security, multithreading, and socket programming. By creating a secure space for users to chat using SSL encryption and synchronized threads, the application effectively meets its design goals. Its simplicity in setup and robustness in functionality make it a strong foundation for further development into a comprehensive, secure messaging platform.

Please open the README file included to find complete installation instructions.

https://github.com/BelaBartok39/Pirate-Chat-Project

References

"Network Programming in Python." n.d. March 2025. <https://bogotobogo.com/python/python_network_programming_server_client.php>.

"TLS/SSL wrapper for socket object." n.d. March 2025. <, https://docs.python.org/2/library/ssl.html>.

Tripathi, Satyam. "Guide to Python Socket Programming." n.d. March 2025. <https://builtin.com/data-science/python-socket>.

Ubah, Kingsley. "Implementing TLS/SSL in Python." n.d. March 2025. <https://snyk.io/blog/implementing-tls-ssl-python/>.

"Wireshark Tutorial and Cheat Sheet." n.d. April 2025. <https://hackertarget.com/wireshark-tutorial-and-cheat-sheet/>.